ᵖᵃ¹

# BEAM: Technology for Autonomous Self-Analysis[1][2]

Ryan Mackey
Mail Stop 126-147
California Institute of Technology
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109
818-354-9659
Ryan.M.Mackey@jpl.nasa.gov

Mark James
Mail Stop 126-347
California Institute of Technology
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109
818-354-8488
Mark.James@jpl.nasa.gov

Han Park
Mail Stop 126-347
California Institute of Technology
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109
818-354-8564
Han.G.Park@jpl.nasa.gov

Michail Zak
Mail Stop 126-347
California Institute of Technology
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109
818-393-5351
Michail.Zak@jpl.nasa.gov

*Abstract*— BEAM (Beacon-based Exception Analysis for Multimissions) is an end-to-end method of data analysis intended for real-time fault detection and characterization. It provides a generic system analysis capability for potential application to deep space probes and other highly automated systems.

This paper describes in brief the architecture, application, and operating theory of BEAM. We will also make reference to companion papers [1] and [2], which describe individual elements of the technology in mathematical detail. BEAM provides a generalized formalism for diagnostics and prognostics in virtually any instrumented system. Consideration is given to all standard forms of data, both time-varying (sensor or extracted feature) quantities and discrete measurements, embedded physical and symbolic models, and communication with other autonomy-enabling components such as planners and schedulers. This approach can be adapted to on-board or ground-based implementations with no change to the basic operating theory. The approach will be illustrated with an overview of application types, past validations, and ongoing efforts.

TABLE OF CONTENTS

## 1. INTRODUCTION

BEAM stands for Beacon-based Exception Analysis for Multimissions, which is a complete method of data analysis for real-time fault detection and characterization. The original intended application of this project was to provide a generic system analysis capability for deep space probes and other highly automated systems. Such systems are typified by complex and unique architectures, high volumes of data collection, limited bandwidth, and a critical need for flexible and timely decision abilities.

Beacon monitoring was the original impetus for this design. Beacon monitoring is a telemetry method wherein a subset of available engineering data is broadcast by the monitored system as follows: Rather than downlink the entire engineering dataset at all times, the approximate condition of the spacecraft or remote system is categorized into one of several "beacon tones." These tones correspond to (a) nominal system operation, (b) anomalous or interesting operation, (c) degradation or significant faults, or (d) total failure. The beacon tone is sent at all times using a much simpler telemetry system. Additional data is sent if and only if specific attention is requested or required of system operators. In other words, beacon monitoring represents a step towards machine self-reliance.

This method is only practicable if a number of hurdles can be overcome. Central to the method is the ability of a spacecraft to perform an accurate, timely, and verifiable

[1] 0-7803-6599-2/01/$10.00 © 2001 IEEE
[2] Updated September 15, 2000

self-diagnosis. Such a self-diagnosis must not only provide a safe operating envelope, but must perform, at worst, comparably to spacecraft experts in a control room. A system that is insensitive, generates false alarms, or requires oversight will not be effective, because such inefficiencies will be amplified in a "streamlined" process.

The basic premise of BEAM is the following: Construct a strategy to characterize a system from all available observations, and then train this characterization with respect to normal phases of operation. In this regard the BEAM engine functions much as a human operator does – through experience and other available resources (known architecture, models, simulation, etc.) an allowed set of behavior is "learned" and deviations from this are noted and examined. Such an approach should be applied as a complement to simplistic but reliable monitors and alarms found in nearly all instrumented systems. The approach should also be constructed such that information products can be used to drive autonomous decisions, or to support the decisions of human operators. In other words, the system must allow for intervention and aid it wherever possible. If this is not done, it is difficult for spacecraft experts to gain trust in the system, and the benefits of beacon operation (or any similar cost-saving approach) will be doomed from the outset. In this manner BEAM is not solely suited to beacon monitoring, but is more broadly applicable to monitored or wholly autonomous systems.

Two important features make BEAM a standout among the various fault-detection technologies that have been advanced. The first is its broad range of applicability. This approach has been used with sensor and computed data of radically different types, on numerous systems, without detailed system knowledge or a priori training. Separate components are included to treat time-varying signals and discrete data, and to interpret the combination of results.

The second is its ability to detect, and with few exceptions correctly resolve, faults for which the detector has not been trained. This flexibility is of prime importance in systems with low temporal margins and those with complex environmental interaction. This ability also comes with few requirements in terms of detector training.

Since its original inception, BEAM has been matured and proven on many separate applications, both on-board and off-board. We will consider the architecture and theory behind this approach and briefly illustrate the implications of such a strategy from these applications.

## 2. Basic Architecture

At the simplest level of abstraction, BEAM is software, which takes data as input and reports fault status as output. Implementation of this software is dependent on the application, but a typical application would have a system with a number of individual components, each of which reports health or performance data to a local computer. At this juncture, a local BEAM manager draws a conclusion based on that data during runtime and forwards the results to a decision maker. We must consider the physical makeup of the device in question when deciding how to compartmentalize the diagnosis. Consideration must be made for computing power, communication and data buses, and the natural divisions present in the system. To accommodate such a wide range of possibilities, the computational engine of BEAM itself is highly adaptable with respect to subsystem size and complexity.

For each single compartment or subsystem, we can expect to receive four types of data:

1. Discrete status variables changing in time – modes, switch positions, health bits, etc. – from sensors or software

2. Real-valued sensor data varying at fixed rates – performance sensors or dedicated diagnostic sensors

3. Command information – typically discrete as in 1.

4. Fixed parameters – varying only when commanded to change but containing important state information

These types of data are all of value but are useful in different ways. Status variables and commands are useful to a symbolic model. Commands and fixed parameters are useful to a physical system model. Time-varying sensor data is useful for signal processing approaches. An optimal strategy must take each of these into account and produce a single, unified decision. In order to study each and combine results, we propose the following BEAM architecture, as presented in Figure 1.

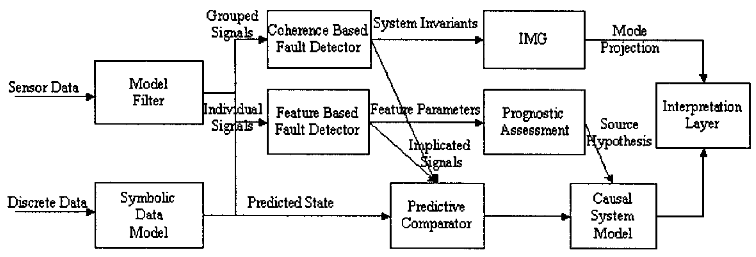

Figure 1: Top-Level BEAM Architecture

A few notes about the architecture are in order before we consider the individual descriptions of its components. Specifically, we should consider the data flow, which is somewhat complicated:

1. Fixed parameters and command information are input to the specific system models (if any). These are contained in the Symbolic Model and the Physical Model components. These data will not be propagated further and influence other components only through the model outputs.

2. Discrete variables will only be propagated through the symbolic components. The symbolic components support the signal processing components by providing a mode determination from the discrete data.

3. Time-varying quantities are separated into two groups as part of the training process. Specifically, signals with high degrees of correlation to others, or those not expected to uniquely indicate a severe fault, are only passed to the Coherence Analysis components. Signals that may uniquely indicate a fault, along with those already flagged as faulty by the coherence analysis, require additional processing and are also passed through the feature extraction components.

4. The split between time-varying signals described in 3. is a computational efficiency consideration and reflects general philosophy of operation, but is not essential. Given adequate resources, there is nothing preventing all time-varying signals from being sent to both types of signal analysis at all times.

Descriptions of the individual components and their internal structure are presented in the following section.

## 3. Component-Level Descriptions

The components of the BEAM architecture are designed to take advantage of different sources of information, and to produce unique conclusions about the health of the system. We must account for all types of data and models, and we wish to perform not only fault detection and isolation, but also novelty detection, prognostic assessment, impact assessment, and diagnostics (implication of malfunctioning components) wherever possible. The various components listed below each perform a particular duty.

### *Model Filter*

The Model Filter, also referred to as the Gray Box, combines sensor data with physical model predictions (run in real-time). We are interested in augmenting sensor data with intuition about the operation of the system. The inclusion of physical models where available is the most efficient means of incorporating domain knowledge into a signal-based processing approach.

The usual methods of signal processing, including the Dynamical Invariant Anomaly Detector component of BEAM, represent "Black Box" strategies; i.e. nothing is known about the internal governing equations of a system. Such linear approaches are effective in general, but there are profound benefits to simultaneous consideration of sensor and physical information. The opposite perspective would be a "White Box" strategy, where a complete and accurate physical simulation was available for comparison to captured data. This case is desirable but rarely practical. In nearly all cases we must make do with a degraded or simplified model, either because of system complexity or computational limits. The "Gray Box" method serves to make use of whatever models are available, as we will briefly explore in this section. For a thorough treatment, please see [1].

Any theoretical dynamical model includes two types of components: those directly describing the phenomena associated with the primary function of the system (such as the effect of torque exerted on the turbine shaft on rotor speed), and those representing secondary effects (such as frictional losses, heat losses, etc.). The first type is usually well understood and possesses a deterministic analytical structure, and therefore its behavior is fully predictable. On the other hand, the second type may be understood only at a much more complex level of description (i.e. at the molecular level) and cannot be simply incorporated into a theoretical model. In fact, some components may be poorly understood and lack any analytical description, such as viscosity of water in microgravity. The main idea of this approach is to filter out contributions that are theoretically predictable from the sensor data and focus on the components whose theoretical prediction is lacking. The filtering is performed by the theoretical model itself.

If we assume that the theoretical model is represented by a system of differential equations, known physical processes will be described with state variables and theoretical functions. But we will also have additional terms that describe phenomena for which a full mathematical description is unavailable or too complex. Examples of this include friction in bearings, material viscosity, and secondary effects such as oscillations or flutter in mechanical systems. These leftover terms represent the unknown space of our partial model.

If we substitute sensor data into the theoretical model, so long as the actual system performs as expected, there will be no departure from the model. However, an abnormality in performance will alter the behavior of the "leftover" terms.

In general, we can treat the abnormality as the result of a stochastic process. If the abnormality is small compared to the modeled components of the system, it will suffice to assign some confidence interval for fault detection. However, if the accuracy of the model is poor, we must treat the leftover terms using stochastic or parameter estimation models, as described in following components.

Compared to a straightforward signal analysis method, where highly stable dynamical models are available, the "black box" approach is not only more laborious, but is also less effective since the stochastic forces become deeply hidden in the sensor data. The practical upshot of the gray box approach is to use the theoretical model as a filter, which damps the deterministic components and amplifies the stochastic components, simplifying the task of fault detection for the other components.

Because this approach can operate upon high- and low-fidelity models, it is highly effective as a means of preprocessing sensor data. Such models are available for the majority of autonomous systems, leftover from the design and analysis efforts to build such systems.

To effectively implement this module, one must have such a design model available, or at the very least an approximate physical understanding of the system behavior. Such models must provide "reasonable" agreement with sensor data, and must therefore output directly comparable quantities at similar data rates. This step requires some human intervention, first to cast (or build) the model in a format to produce such results, and second to verify the model's fidelity against a known standard – be that a superior model, an engineering model, or prototype or actual flight data. While the model need not be of precise fidelity to be useful, it is important to confirm the stability of the model and its comparability to sensor information. Use of a poor model will increase the reliance upon signal-based methods downstream, or in extreme cases destabilize the method entirely, in which case only signal-based methods may be applied to the sensor data.

### *Symbolic Data Model*

In the overall BEAM strategy, real-time measurements are combined with predicted and expected behavior along with predicted performance to quickly isolate candidate faults. The Symbolic Data Model (SDM) is the first line of defense in determining the overall health of the system and it is the primary component that determines its active and predicted states. It operates by examining the values from status variables and commands to provide an accurate, evolving picture of system mode and requested actions. The overall approach afforded by BEAM extends considerably beyond more conventional symbolic reasoning. Since most rule-based diagnostic systems (expert systems) provide only this module and nothing else, they are limited in that they can only identify and diagnose anticipated problems.

Knowledge in the SDM is represented as rules, which are themselves composed of patterns. The rule is the first Aristotelian syllogism in the form: ***If* ... *Then*...** .The variables of the syllogism are joined by the ***And/Or*** logical connections. The selector ***Else*** points to other cases. This formula is a rule; the rules are sequenced in the succession of logical thinking or pointed at a jump in the sequence (**Else -> Go To**).

Patterns are relations that may or may not have temporal constraints, i.e., may only hold true at certain times or persist for the entire duration of the analysis. Patterns define the constraints that must hold true in order for the antecedent to succeed.

Conceptual representation is the main way to formulate the patterns of the system as part of a computer program. The essential tool the SDM uses is a rule; that is the reason why expert systems can also be called rule-based systems.

The SDM operates by using many small slivers of knowledge organized into conditional If-Then rules. These rules are then operated on in a variety of different ways to perform different reasoning functions.

Unlike the numeric models, the SDM requires a knowledge base in order to perform its analysis functions. From several points of view, representation of knowledge is the key problem of expert systems and of artificial intelligence in general. It is not by chance that the term "knowledge-based systems" has been applied to these products.

The generic features of knowledge are embodied in this representation. The domain expert stores the objects, actions, concepts, situations and their relations using the SHINE (Spacecraft High-speed Inference Engine) [3] representation language and this is stored in the SDM knowledge base. The collection of this knowledge represents the sum total of what the SDM will be able to understand. The SDM can only be as good as the domain expert that taught it.

The SDM generates two primary kinds of results: derived states and anomalies. To provide a uniform representation, we use the identical approach in performing each of these functions and they differ only in their knowledge bases that they use. Derived states are sent on to signal-processing components as well as other discrete components. Anomalies, as the name implies, are concrete indications of faults.

### *Coherence Based Fault Detector*

The Coherence Based Fault Detector is a general method of anomaly detection from time-correlated multi-signal sensor data. This is described mathematically in [2] along with an illustrative application. The method is applicable to a broad class of problems and is designed to respond to any departure from normal operation, including faults or events that lie outside the training envelope.

Also referred to as the SIE (System Invariance Estimator), it receives multiple time-correlated signals as input, as well as a fixed invariant library constructed during the training process (which is itself data-driven using the same time-correlated signals). It returns the following quantities:

- Mode-specific coherence matrix
- Event detection
- Comparative anomaly detection
- Anomaly isolation to specific signals
- Distance measure of off-nominal behavior

As a first step of analysis, this computation makes a decision whether or not a fault is present, and reduces the search space of data to one or a few signals. Time markers are included to indicate the onset of faulted data. These conclusions, which can be drawn for nearly any system, are then passed to other analysis components for further feature extraction, correlation to discrete data events, and interpretation.

To motivate a cross-signal approach, consider that any continuously valued signal, provided it is deterministic, can be expressed as a time-varying function of itself, other signals, the environment, and noise. The process of identifying faults in a particular signal is identical to that of analyzing this function. Where the relationship is constant, i.e. follows previous assumptions, we can conclude that no physical change has taken place and the signal is nominal. However, the function is likely to be extremely complex and nonlinear. Environmental variables may be unmeasurable or unidentified. Lastly, the interaction between signals may be largely unknown. For this reason it is more efficient to study invariant features of the signals rather than the entire problem.

Because we do have the different signal measurements available, we can consider relationships between signals separately and effectively decouple the problem. A good candidate feature is signal cross-correlation. By studying this or a similar feature rather than the raw signals, we have reduced our dependence on external factors and have simplified the scope of the problem.

In the case of the SIE we will use a slightly different feature across pairs of signals, which we refer to as the coherence coefficient. It is chosen instead of the ordinary coefficient of linear correlation in order to take advantage of certain "nice" mathematical properties. This coefficient, when calculated for all possible pairs of N signals, describes an NxN matrix of values. The matrix is referred to as the Coherence Matrix of the system.

The coherence matrix, when computed from live streaming data, is an evolving object in time with repeatable convergence rates. Study of these rates allows us to segment the incoming data according to mode switches, and to match the matrix against pre-computed nominal data.

For the purpose of this discussion, a "Mode" refers to a specific use or operation of the system in which the coherence coefficients are steady. In other words, the underlying physical relationships between parameters may change, but should remain constant within a single mode. These modes are determined from training data for the purpose of detector optimization. Ordinarily they do correspond to the more familiar "modes," which represent specific commands to or configurations of the system, but they need not be identical. Frequently such commands will not appreciably alter the physics of the system, and no special accounting is needed.

Comparison of the runtime coherence matrix to a pre-computed, static library of coherence plots, taking into account the convergence behavior of the computation, is an effective means of anomaly detection and isolation to one or more signals.

Unfortunately, this comparison is only meaningful if we can guarantee our present coherence values do not reflect mixed-mode data, and so some method of segmentation must be found. For purposes of anomaly detection, mode boundaries can be detected by monitoring the self-consistency of the coherence coefficients. As each new sample of data is included into the computation, a matrix average for the resulting change is extracted and compared against the expected convergence rate. A change in the convergence rate implies a new mode has been entered and the computation must be restarted.

Between detected mode transitions, the difference between the computed and expected coherence allows us to optimally distinguish between nominal and anomalous conditions. Violation of this convergence relationship indicates a shift in the underlying properties of the data, which signifies the presence of an anomaly in the general sense. The convergence rate of this relationship, used for fault detection, is considerably slower than that for data segmentation, though still fast enough to be practical.

Once a fault has been indicated, the next step is to isolate the signals contributing to that fault. This is done using the difference matrix, which is formed from the residuals following coherence comparison against the library.

Because nearly every autonomous system relies upon performance data for operation as well as fault protection, this method is applicable to a wide variety of situations. The detector increases in accuracy as the number of sensors increases; however, computational cost and mode complexity eventually place a practical limit on the size of the system to be treated. At the extremes, this method has been successfully applied to systems as small as four sensors and as complex as 1,600 of radically varying type.

The analysis involves computation of an NxN matrix, and therefore the computation scales quadratically with the number of signals; fortunately, most designs are implicitly hierarchical, allowing the problem to be separated if computational costs are too great. Furthermore, typical systems or subsystems exhibit a reasonable number of sensors for this method given the state-of-the-art in flight processors. A real-world example of this technique using a single embedded flight processor (PowerPC clocked at 200 MHz) demonstrated real-time capability on a 240-sensor system sampled at 200 Hz.

Another key virtue of this approach is its resilience in the face of novelty. The coherence between signals is a very repeatable property in general, especially as compared to environmental variable or nonlinear terms in the signals themselves. This repeatability allows us to quickly determine whether or not the coherence is consistent with any of the training data, and therefore can be used as an efficient novelty detector, regardless of its cause.

Training of this component is entirely data-driven. Related paper [2] demonstrates the training process. In order to be most effective, the component should be trained with examples of nominal operating data covering every major mode of operation. These examples should be sufficiently detailed to capture the expected system dynamics. While the actual training of the detector is completely autonomous once the task of separating nominal and anomalous data has been performed, this task – and the collection of the data – can be difficult.

In cases where the training data is difficult to obtain or identify, the component functions best in a "learning" mode, similar to its performance following anomaly detection. If we expect to see novel data that does not indicate faults, we must provide for a feedback to the detector, which is a human-intensive process. Novel data can be tagged by the detector and archived for study. Following classification as nominal or anomalous, the detector can "retrain" using this data and continue. This technique has been used effectively in the study of untested systems.

In cases where sensor data is relatively isolated, or when sample rates preclude the resolution of system dynamics, this method is not likely to be effective. These cases place a much greater reliance upon the symbolic method components of BEAM.

*Dynamical Invariant Anomaly Detector*

The Dynamical Invariant Anomaly Detector is designed to identify and isolate anomalies in the behavior of individual sensor data. Traditional methods detect abnormal behavior by analyzing the difference between the sensor data and the predicted value. If the values of the sensor data are deemed either too high or low, the behavior is abnormal. In our proposed method, we introduce the concept of *dynamical invariants* for detecting structural abnormalities.

Dynamical invariants are governing parameters of the dynamics of the system, such as the coefficients of the differential (or time-delay) equation in the case of time-series data. Instead of detecting deviations in the sensor data values, which can change simply due to different initial conditions or external forces (i.e. operational anomalies), we attempt to identify structural changes or behavioral changes in the system dynamics. While an operational abnormality will not lead to a change in the dynamical invariants, a true structural abnormality will lead to a change in the dynamical invariants. In other words, the detector will be sensitive to problems internal to the system, but not external disturbances.

We start with a description of a traditional treatment of sensor data given in the form of a time series describing the evolution of an underlying dynamical system. It will be assumed that this time series cannot be approximated by a simple analytical expression and does not possess any periodicity. In simple words, for an observer, the future values of the time series are not fully correlated with the past ones, and therefore, they are apprehended as random. Such time series can be considered as a realization of an underlying stochastic process, which can be described only in terms of probability distributions. However, any information about this distribution cannot be obtained from a simple realization of a stochastic process unless this process is stationary -- in this case, the ensemble average can be replaced by the time average. An assumption about the stationarity of the underlying stochastic process would exclude from consideration such important components of the dynamical process as linear and polynomial trends, or harmonic oscillations. Thus we develop methods to deal with non-stationary processes.

Our approach to building a dynamical model is based upon progress in three independent fields: nonlinear dynamics, theory of stochastic processes, and artificial neural networks.

After the sensor data are stationarized, they are fed into a memory buffer, which keeps a time history of the sensor data for analysis. We will study critical signals, as determined by the symbolic components of BEAM, the operating mode, and the cross-signal methods outlined above. The relevant sensor data is passed to a Yule-Walker parameter estimator. There, the dynamical invariants and the coefficients of the time-delay equation are computed using the Yule-Walker method.

Once the coefficients are computed, they will be compared to the ones stored in a model parameter database. This contains a set of nominal time-delay equation coefficients appropriate for particular operating mode. A statistical comparison will be made between the stored and just-computed coefficients using a bootstrapping method, and if a discrepancy is detected, the identity of the offending sensor will be sent on.

Further analysis is carried out on the residual or the difference between the sensor data values and the model predicted values, i.e. the uncorrelated noise, using a nonlinear neural classifier and noise analysis techniques. The nonlinear neural classifier is designed to extract the nonlinear components, which may be missed by the linear Yule-Walker parameter estimator. The weights of the artificial neural network, another set of dynamical invariants, will be computed and compared with nominal weights stored in the model parameter database. Similarly, the noise characteristics, such as the moments of probability distribution, are dynamic invariants for stationarized sensor data, and will be compared with those stored in the Model Parameter Database. If any anomalies are detected in either the nonlinear components or the noise, the identity of the sensor will be sent to the channel anomaly detector.

Finally, the channel anomaly detector aggregates information from the Yule-Walker parameter estimator, nonlinear neural classifier, and noise analysis modules, and classifies the anomaly before sending the fault information to the Predictive Comparison module, which is discussed below.

Like the SIE described above, training of this detector is data-driven. It has similar requirements in terms of data set and human involvement. Also like the SIE, insufficient data training will result in false alarms, indicating novelty, until data collection and review during flight operations produce a sufficiently large data set to cover all nominal operating modes.

Also like the SIE, this method is only likely to be effective if system dynamics are captured in the sensor data. However, this method is effective on isolated sensors, though it is often not sensitive to interference or feedback faults that manifest on no particular sensor.

*Informed Maintenance Grid (IMG)*

The purpose of the Informed Maintenance Grid (IMG) is to study evolution of cross-channel behavior over the medium- and long-term operation of the system. Tracking of consistent deviations exposes degradations and lack of performance.

The IMG itself is a three-dimensional object in information space, intended to represent the evolution of the system through repeated use. The IMG is constructed from results from the SIE described above, specifically the deviations in cross-signal moments from expected values, weighted according to use and averaged over long periods of operation. The cross-signal moments are a persistent quantity, which amplifies their value in long-term degradation estimation.

There are two convincing reasons to consider cross-signal residuals in this fashion. First is the specific question of degradation and fault detection. Degradation typically manifests as a subtle change in operating behavior, in itself not dramatic enough to be ruled as a fault. This emergent behavior frequently appears as subthreshold residuals in extracted signal features. As described above, statistical detection methods are limited in sensitivity by the amount of data (both incident and training data) that can be continuously processed. However, tracking of consistent residuals over several such experiments can lead to a strong implication of degraded performance.

The second reason addresses functional health of the system. In addition to tracking the magnitude and consistency of residuals arising from degradation, we can also observe their spread to other signals within the system and track their behavior relative to different modes of usage.

The IMG produces a long-term warning regarding system-wide degradations. This differs slightly from the companion Prognostic Assessment module, which concerns itself with individual signals and preestablished operating limits. However, the IMG is also useful with regard to novel degradations. Such are the norm with new advanced systems, as predicting degradation behavior is very difficult and much prognostic training must be done "on the job."

Visually, the three-dimensional object produced by the IMG is an easily accessible means of summarizing total system behavior over long periods of use. This visual means of verifying IMG predictions makes BEAM easily adapted for applications with human operators present.

*Prognostic Assessment*

The Prognostic Assessment component provides a forward-projection of individual signals using their model parameters. Based upon this, it establishes a useful short-term assessment of impending faults.

The channel level prognostics algorithm is intended to identify trends in sensor data which may exceed limit values, such as redlines. This by itself is a common approach. However, given the richness of feature classification available from other BEAM components, it is highly effective to update this procedure. A stochastic model similar in principle to the auto-regressive model is used to predict values based upon previous values, viz. forecasting. The aim is to predict, with some nominal confidence, if and when the sensor values will exceed its critical limit value. This permits warning of an impending problem prior to failure.

In general, time-series forecasting is not a deterministic procedure. It would be deterministic only if a given time series is described by an analytical function, in which case the infinite lead time prediction is deterministic and unique based upon values of the function and all its time derivatives at $t = 0$. In most sensor data, this situation is unrealistic due to incomplete model description, sensor noise, etc. In fact, present values of a time series may be uncorrelated with previous values, and an element of randomness is introduced into the forecast.

Such randomness is incorporated into the underlying dynamical model by considering the time series for $t \leq 0$ as a realization of some (unknown) stochastic process. The future values for $t > 0$ can then be presented as an ensemble of possible time series, each with a certain probability. After averaging the time series over the ensemble, one can represent the forecast as the mean value of the predicted data and the probability density distributions.

The methodology of time series forecasting is closely related to model fitting and identification. In general, the non-stationary nature of many sensor data may lead to misleading results for future data prediction if a simple least-square approach to polynomial trend and dominating harmonics is adopted. [4] The correct approach is to apply inverse operators (specifically difference and seasonal difference operators) to the stationary component of the time series and forecast using past values of the time series.

To implement this module, we begin by feeding a Predictor stationary data, the auto regressive model coefficients, past raw data values, and limit values; i.e., everything required to evaluate the prediction plus a redline value at which to stop the computation. The predictor will generate many predictions of time to redline and pass them on to the Redline Confidence Estimator. The Redline Confidence Estimator will then construct a probability distribution of the time when the channel value will exceed the redline limit. Finally, the Failure Likelihood Estimator takes the probability distribution and computes the likelihood (probability) that the channel value may exceed the redline value within some critical time. If the probability exceeds a certain preset threshold as determined by the application, e.g. 99% confidence, then the critical time and its probability will be sent to the symbolic components.

Implementation of this Prognostic Assessment is relatively easy provided guidelines exist about the allowed performance of each parameter. The model coefficients are automatically sensed from data, leaving only the selection of a threshold – usually determined by control functions – to be determined.

### *Predictive Comparator*

The Predictive Comparison (PC) component compares the requested and commanded operation of the system versus the sensed operation as interpreted from the time-varying quantities. Its goal is to detect misalignment between system software execution and system hardware operation. This is a principal concern, as we are dealing with systems that rely on a large degree of software control, if not complete autonomy.

The PC combines the results from the numeric and symbolic engines and looks for confirmation and differences between them. It is the primary interface that merges the symbolic results for the system predicted state and explicit failures with suspected faulty channel implications and event detections from the signal-based components. Its result is a sequence of confirmed predicted failures and detected unmodeled events.

A failure is considered confirmed when both the numeric and symbolic engines each predict the same failure or system state change. Unmodeled events are cases where the numeric and symbolic engines differ in their conclusions or where no conclusion can be reached based on the training information. The capability to compare parallel analysis engines, each approaching the problem from an entirely different theoretical foundation, is a particularly asset of BEAM.

This module uses generic symbolic processing algorithms and does not require a knowledge base in order to perform its function. The following kinds of comparisons are made:

1. Changes in the system predicted state from the symbolic engine are correlated to detected events from the numeric engine. If the numeric engine generates an event and it approximately correlates with a predicted state change, then the predicted state is considered confirmed.

2. Signals that are implicated as failed by the symbolic engine are correlated to signals implicated by numeric engine. When there is agreement, the channel is confirmed as faulty. When there is a difference between the two, the signal is marked as an unmodeled event.

The final component in the PC is to merge results from items 1 and 2 with the list of explicit failures and events so multiple redundant conclusions of bad signals and unmodeled events are not generated.

### *Causal System Model*

The Causal System Model (CSM) is a connectivity matrix designed to improve source fault isolation and actor signal identification. In the SDM, the entire domain knowledge is represented as If-Then rules only. When the domain is very large and complex, an entirely rule-based representation and associated inference leads to a large and inefficient knowledge base, causing a very poor focus of attention. To eliminate such unwieldy knowledge bases in the SDM engine, we provide a causal system model. This component simplifies the problem by looking for relationships between observations in the data to fill in blanks or gaps in the information from the SDM.

The CSM accomplishes this by decomposing the problem into smaller modules, called knowledge sources, and by providing a dynamic and flexible knowledge application strategy. The same concept can be extended to problems requiring involvement of multiple agents representing different domains.

The CSM reacts as and when conflicts arise during problem solving and uses conflict-resolution knowledge sources in an opportunistic manner. Essentially, the CSM provides a high-level abstraction of knowledge and solution and the derived relationships between observation and implication.

The three basic components of the CSM are the knowledge sources, blackboard data structure, and control. In the CSM, knowledge required to solve the problem is decomposed into smaller independent knowledge sources. The knowledge sources are represented as SHINE If-Then rules. Each rule set or knowledge source contains knowledge to resolve one task in the diagnostic model. The blackboard holds the global data and the information on the problem-solving states. Activation of the knowledge sources modifies the states in the blackboard leading to an incremental causal relationship for actor signal identification.

Since the causal relationship is decomposed into a hierarchical organization, the concept of an event becomes predominant in this blackboard-centered strategy. Any change in the blackboard is considered an event. Any change in the solution state either due to generation of additional information or modification of existing is immediately recorded. The execution controller notes this change and takes the necessary actions by invoking an appropriate knowledge source. This process repeats until the final causal relationship is obtained.

Since the CSM is fed with a real-time stream of events (anomalies, suspected bad signals, events and unmodeled events), the arrival of a new event can make a previous concluded causal relationship incorrect. In such cases, corresponding stages have to be undone by backtracking all the previously made assumptions leading to the reasoning to be non-monotonic. This requires a dependency network to be incrementally maintained as the causal assumptions are generated using the knowledge sources.

*Interpretation Layer*

The Interpretation Layer (IL) collates observations from separate components. It submits a single fault report in a format usable to recovery and planning components (in the case of a fully autonomous system) or to system operators. This is a knowledge-based component that is totally dependent upon the domain and the desired format of the output.

As its inputs it accepts a list of events from the CSM and possible conclusions from the SDM as described above. Any supported conclusion that the SDM generates is considered a final output and is translated into the required output format.

The CSM events can be decomposed into rule-based anomalies and detected events from the numeric engine. The interpretation layer performs a many-to-many mapping of faults (events) to interpretations. Each interpretation has a context associated with it. Because of this context, when multiple interpretations are generated within the same context, they can be grouped together as one interpretation containing several elements. This is typical of events in complex systems in general.

Contexts are assigned by a contextual engine within the interpretation layer. Its purpose is to look for commonalties among each unique interpretation. In this manner if there is either a causal or interdependent relationship between interpretations, they are considered as possibly related. For example, if we have an alarm condition occurring on signals monitoring volts and/or amps, and the SDM concluded a fault based upon the number of watts generated, the engine will combine the volts and amps alarms with the watts conclusion. This provides for a very concise statement of the fault at hand without the user being deluged with disjointed information from many different sensors.

The final reduced set of interpretations is processed by a component that reduces interpretations to conclusions. A rule-based model is used to apply relationship definitions between interpretations, their causal relationships and their supported conclusion. For example, if the SDM did not generate a conclusion for watts being in alarm based upon the signals of volts and amps being overranged, then such a conclusion can be made here and generated as a final output.

## 4. APPLICATIONS

BEAM has been studied on numerous applications and is currently being formulated for inclusion into a number of systems, some of which are described in related papers. In this section we will consider the basic classes of application and the value of such a sensing strategy. The shape and scope of BEAM is highly dependent on the application, according not only on the specific system, but on its mission plan as a whole. As such, it is best described by example, but specifics of this nature go beyond the scope of this high-level survey paper. Please see the related references for more in-depth analysis of particular cases, which highlight certain qualities of BEAM.

*Telemetry Limited Remote Systems*

The first and most obvious customer for BEAM is typical of systems developed at JPL. The core mission of JPL is robotic space exploration. Such systems are usually managed by a team of experts who review the engineering data on a regular basis. These missions are also very expensive, and represent a considerable investment in manpower and time.

Cases like these, where we can count on human experts to oversee the analysis process, are well suited for BEAM. In these cases BEAM may serve to enhance operator effectiveness by providing a first look at the data, effectively compressing it to contain only the relevant, "most interesting" parts. Benefits as a result can be expressed in terms of workload, safety, turnaround time, reliability, or cost in work-hours.

Complex spacecraft are almost always accompanied by accurate models and simulation left over from the design process. They undergo a rigorous qualification and are tested through a variety of fault cases. However, they are impossible to repair or upgrade (except through software), they are typically unique, and because of this and the unique environments they encounter, they face a large number of "novel" situations.

There are two basic philosophies to BEAM for spacecraft applications: on-board and off-board. Off-board implementation provides operators with a simple tool to take full advantage of design models and to study systemwide interactions. It isolates all incidences of "failed" and "interesting" data, and for these cases produces a best-guess of the source. These results are logged for future comparison and long-term degradation analysis.

On-board application is more difficult and less flexible, owing to more rigorous software qualification and more stringent processing requirements. However, there are specific benefits that can only be realized in an on-board implementation. Most important is the time-criticality of detection. Faults can interrupt critical phases of deep space missions, such as spacecraft maneuvering or encounter. Because of the large delays in communication, it is highly desirable to provide detection and recovery capabilities to handle a wide variety of unforeseen occurrences.

In both cases, BEAM functions to reduce operator workload and mission cost while providing quicker spacecraft understanding and broader safety margins. On-board implementation offers a further increase in system safety, but is much less tolerant of false alarms. When operated strictly off-board, much higher sensitivity may be requested by the operators to aid in their analysis of special cases.

Experiments using BEAM in both configurations were conducted using the Cassini spacecraft Attitude and Articulation Control Subsystem (AACS) simulation (on-board analysis) and actual flight data (off-board analysis). The AACS is an extremely complex subsystem, incorporating approximately 1,600 possible telemetry signals at rates of up to 8 Hz.

Results of these analyses were presented at the AIAA '98 conference in Huntsville, Alabama [5]. For the on-board simulation case, BEAM was able to match the performance of existing fault detection software – and in some cases exceed it, particularly with respect to time of detection -- over a portion of the fault protection test grid. The off-board analysis experiment was a blind study applied to the first six months of flight data, including pre-flight, launch, and systems checkout.

Despite lacking the commands sent to the spacecraft and knowledge of the system design, BEAM was able to automatically detect and isolate all unusual events. This was compared against the Incident / Surprise / Anomaly (ISA) reports, produced by the operations team, at the end of the experiment. Such a result means "no false negatives," i.e. no missed detections, were produced by the detector.

With regard to false positives, the question of performance is more complicated. In addition to replicating the ISA reports both in terms of (a) detection, (b) timing, and (c) affected signals, BEAM also detected three events not reported in the ISA logs. These three events were later revealed to be (1) launch of the spacecraft, (2) first trajectory correction maneuver (TCM), and (3) second TCM. However, given that the BEAM system was unaware of the commands sent to the spacecraft, this result is neither surprising nor troublesome. Cassini is an example of a unique system with a large amount of ground support, and is therefore relatively tolerant of false positives. Such is not the case for many of the examples described below.

*Reliability Centered Control*

A very different class of problem is faced by many earth-bound systems that nonetheless can benefit from this approach. There are numerous examples of complex machinery that rely upon internal sensing and expert operators to provide operating margins for reasons of safety or efficiency. In such an application, like the spacecraft example above, pure autonomy is not necessary. Instead it is desirable to empower the system operators by making system health management an easily accessible and controllable function.

This problem is typified by the Deep Space Network (DSN), which is managed by JPL. Downlink is a crucial element in space exploration, as spacecraft observations are useless if their results cannot be retrieved. The communications antennae are highly subscribed, making reliable operation of great importance. DSN antennae are complex electromechanical structures, in many ways no different from spacecraft or aircraft systems. They contain power supplies, hydraulics, signal generators, radiating elements, and so on. Many of these systems are constructed with health and performance monitors.

BEAM integration with DSN console tools is an ongoing task at JPL. In particular, we seek to allow control of an entire array of communications antennae, including the ability to "hot swap" other antennae if constraints or faults affect the system, from a single console.

Preliminary results of this effort were published in [6]. Work is ongoing to mature this system, and to provide direct communication with automated planners and other components [7].

*Maintenance Cost Reduction*

A third class of problem applies to domains such as piloted aircraft. In such an application, the focus is not on producing new tools for the pilot, but instead upon simplifying the job of maintaining the system.

Aircraft maintenance is a costly endeavor, often involving scheduled maintenance and inspections in addition to replacement of faulty components, which is made more costly by high false alarm rates and poor isolation. Additionally, large fleets of aircraft require relatively simple repair procedures, as system experts are usually unavailable in quantity. This differs greatly from the previous cases. We wish to take advantage of the large body of work and data existing for such aircraft, and their greater numbers, to drive a system that makes analysis and repair a routine task. Furthermore, since the fleet of aircraft is expected to age and degrade with time, we require tools to detect and characterize these new phenomena to allow meaningful prognostics.

A good example of this technology applied to a current aircraft system is given in [2]. This specific example shows how an aircraft hydraulic system can be treated using this method. The result is a diagnostic tool that is very quickly trained, enjoys excellent false-alarm rates, and exceeds the performance of current-day fault detection techniques.

*Full Autonomy*

The last and most complicated class of application concerns a totally autonomous system. This can be thought of as an extreme case of any of the previous examples – for instance, a deep space probe with its own sequence-generation capability, or a pilotless aircraft.

The prospect of a UAV (Unmanned Aerial Vehicle) carries with it some special concerns. For such an application, advanced fault detection and isolation is integral to many types of constraint management. In addition to the ordinary problem of maintenance and safety, system health plays an important role in determining mission capability, and the complex environmental interaction and vehicle performance must be watched closely. Above and beyond all of this are the problems of safety certification and shared airspace. Any self-monitoring technology will be stretched beyond the ordinary task of monitoring hardware to include observations on the software performance as well.

BEAM is a major component of the REAC proposal, referenced in [8] and [9]. REAC seeks to provide an integrated suite of autonomy components designed to permit efficient UAV operation in the face of these concerns. BEAM is particularly effective in this role for two reasons. First is the ability to detect and characterize novelty, which greatly simplifies corrective actions and expands safety margins. Second is the sophisticated integration of signal-based and symbolic analysis. This capability allows BEAM to consider the total state of the aircraft, including performance data as well as commands, constraints, and sensor data in a coherent fashion.

## 5. CONCLUSION

Presented here is the framework for BEAM, a comprehensive self-analysis tool suitable for inclusion for virtually any monitored system. As the aerospace industry faces greater challenges in mission complexity, advanced design, aggressive cost targets, and true autonomy, so must we evolve the support structure that goes with them. In most cases, system modeling and available sensors are more than adequate for the task. Yet it remains to take full advantage of this information. Extracting knowledge from the data and domain information is an essential step in the process of self-analysis and control.

It is the opinion of the authors that a strategy for autonomous control must include all sources of information, and that this information must be fused methodically and deep within the process in order to be of greatest use. The method outlined in this paper seeks to engage all such data at the subsystem level, and from it construct simple but profound conclusions as to the operating health and capability of the system, as well as projected viability and corrective actions.

No such strategy can ignore the fundamental question of feasibility, no matter how attractive. For this reason our system seeks to mimic the logic of a human operator, and draws its training from many of the same sources. The fault detection, isolation, and prognostic conclusions are based upon physical models of arbitrary fidelity, symbolic models, example nominal data, real-time data, and architectural information such as connectivity and causal diagrams. The algorithms have also been scaled to operate comfortably on current-generation flight processors.

Such an architecture makes adaptability to numerous systems possible, but also increases the usefulness of the system in concert with more standard methods of control and analysis. Because the data products themselves are accessible at every stage of analysis, BEAM can benefit systems run at any stage of autonomy -- from the extreme of complete self-determination as part of the core flight software, to the opposite of complete human control as a highly advanced analysis tool.

This last point is more valuable than it may appear. A simple reality of advanced systems is that no testing process can ever be complete, and any new system faces considerable uncertainty in the field. It is a given that novelty will be encountered, and in order to improve upon the system, its safeguards must allow it to survive, to characterize the novelty, and to permit improvements to its software control. This process, in nearly all cases, requires human intervention. BEAM approaches this problem by incorporating physics models and studying embedded features of the operating physics of the system in order to isolate a broad class of anomalies, and applies all available rules and limits of allowed system performance. Following isolation, the results of this analysis are communicated to system experts, who may then revise the software through modifications to those same models, inclusion of new training data, or addition of new symbolic rules.

This process, begun at JPL as a method of improving spacecraft safety and operating costs, is presently under study or being applied to numerous aerospace applications, across the entire spectrum of autonomy. It is hoped that this work will contribute to the greater problem of flight software in total, and thereby simplify some of the challenges facing the next steps in vehicle autonomy.

## ACKNOWLEDGEMENT

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

## REFERENCES

[1] M. Zak and H. Park, "Gray Box Approach to Fault Diagnosis of Dynamical Systems," *The 2001 IEEE Aerospace Conference*, Big Sky, Montana, March 2001.

[2] R. Mackey, "Generalized Cross-Signal Anomaly Detection on Aircraft Hydraulic System," *The 2001 IEEE Aerospace Conference*, Big Sky, Montana, March 2001.

[3] M. James and D. Atkinson, "Software for Development of Expert Systems," *NASA Tech Briefs*, Vol. 14, No. 6, June 1990.

[4] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, *Time Series Analysis*, New Jersey, Prentice Hall, 1994.

[5] S. Gulati and R. Mackey, "BEAM: Autonomous Diagnostics and Prognostics for Complex Spaceborne Systems," (poster session) AIAA 1998, Huntsville, Alabama, November 1998.

[6] M. James and L. Dubon, "An Autonomous Diagnostic and Prognostic Monitoring System for NASA's Deep Space Network", *The 2001 IEEE Aerospace Conference*, Big Sky, Montana, March 2001.

[7] A. Fukunaga, G. Rabideau, S. Chien, and D. Yan, "Toward an Application Framework for Automated Planning and Scheduling," *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation for Space*, Tokyo, Japan, July 1997.

[8] R. Colgren, S. Gulati, and R. Koneck, "Technologies for Reliable Autonomous Control (TRAC)," *IASTED '99 Control and Applications Conference*, Banff, Canada, July 1999.

[9] P. Schaefer et. al., "Technologies for Reliable Autonomous Control (TRAC) of UAVs," *The 2001 IEEE Aerospace Conference*, Big Sky, Montana, March 2001.

**Ryan Mackey** received his B.A. degree from the University of California at Santa Cruz (1993) for Mathematics and Physics, and went on to an M.S. (1994) and Eng. (1997) in Aeronautics at Caltech. He is presently a senior researcher and charter member of the Ultracomputing Technologies Research Group at the Jet Propulsion Laboratory. His research centers upon revolutionary computing methods and technologies for advanced machine autonomy, specifically deep space missions, UAVs and maintainable aerospace vehicles. His interests also include quantum- and biologically-inspired computing.

**Mark L. James** is a senior researcher of the Ultracomputing Technologies Research Group at Jet Propulsion Laboratory. At JPL he is Principal Investigator in real-time inference and knowledge-based systems. His primary research focus is on high-speed inference systems and their application to planetary and deep spacecraft systems. His expertise includes core artificial intelligence technology, high-speed real-time inference systems and flight system software architectures. He has received a number of NASA awards that include NASA Software of the Year Award Nominee.

**Michail Zak** is a senior research scientist in the Ultracomputing Technologies Research Group at the Jet Propulsion Laboratory, California Institute of Technology. He has been with JPL since 1977. His research interests include nonlinear dynamical system theory, chaos theory, quantum information processing, neural networks, and complex systems theory. He introduced the new concept of terminal attractors/repellers in the theory neurodynamics for neural networks, and recently introduced the concept of quantum recurrent nets for computing by quantum simulation. He is the author of over 150 scientific and technical papers, and research monographs.

**Han Park** is a member of the technical staff in the Ultracomputing Technologies Research Group in the Information and Computing Technologies Research Section at the Jet Propulsion Laboratory, California Institute of Technology. He joined JPL in 1999 and performs research and development of fault detection/diagnosis algorithms for aircraft and spacecraft systems. He received his B.S. in M.E. at the University of California at Berkeley, S.M. in M.E. at MIT, and Ph.D. in Aeronautics at the California Institute of Technology. His research interests are in the areas of vehicle health monitoring, signal processing, image processing, pattern recognition, color recognition, quantitative visualization, fluid mechanics, and heat transfer.